VIETNAM GENERAL CONFEDERATION OF LABOUR
**TON DUC THANG UNIVERSITY**
**FACULTY OF INFORMATION TECHNOLOGY**


**VO NHAT HAO**
**PHAM VAN PHUC**

# MIDTERM PROJECT

# INTRODUCTION TO MACHINE LEARNING

# SOFTWARE ENGINEERING

**HO CHI MINH CITY, YEAR 2025**

VIETNAM GENERAL CONFEDERATION OF LABOUR
**TON DUC THANG UNIVERSITY**
**FACULTY OF INFORMATION TECHNOLOGY**


**VO NHAT HAO – 522H0090**
**PHAM VAN PHUC – 522H0068**

# MIDTERM PROJECT

# INTRODUCTION TO MACHINE LEARNING

# SOFTWARE ENGINEERING

**Prof., Dr. LE ANH CUONG**

**HO CHI MINH CITY, YEAR 2025**

# ACKNOWLEDGMENT

- We sincerely thank Mr. Le Anh Cuong for her constant support and enthusiastic direction throughout our investigation and final report.

- We also like to thank Ton Duc Thang University’s Faculty of Information Technology for providing us with an enriching academic environment. The faculty's willingness to share vital expertise and reference materials has not only aided our research endeavor, but has also improved our overall educational experience at the university.

- As we wrap up our study project, we reflect on the vital lessons and insights learned from our educators. Regardless of our limitations and areas for improvement, we are willing to learn and grow. We really seek further assistance to improve our work and appreciate the critical input from our professors and classmates. With their continuous assistance, we are determined to improve our research talents in future initiatives.

- We wish all of our teachers and friends ongoing health and happiness, as their support and care have been invaluable to us on our path.

*Ho Chi Minh City, day 11 month 03  year 2025*
*Author*

*Vo Nhat Hao*

*Pham Van Phuc*

This thesis was carried out at Ton Duc Thang University.

Advisor: .....

.....

*(Title, full name and signature)*

This thesis is defended at the Undergraduate Thesis Examination Committee was hold at Ton Duc Thang University on … /…/……

Confirmation of the Chairman of the Undergraduate Thesis Examination Committee and the Dean of the faculty after receiving the modified thesis (if any).

| **CHAIRMAN** | **DEAN OF FACULTY** |
| --- | --- |
| **……** | **……** |

# DECLARATION OF AUTHORSHIP

I hereby declare that this thesis was carried out by myself under the guidance and supervision of Le Anh Cuong and that the work and the results contained in it are original and have not been submitted anywhere for any previous purposes. The data and figures presented in this thesis are for analysis, comments, and evaluations from various resources by my own work and have been duly acknowledged in the reference part.

In addition, other comments, reviews and data used by other authors, and organizations have been acknowledged, and explicitly cited.

**I will take full responsibility for any fraud detected in my thesis**. Ton Duc Thang University is unrelated to any copyright infringement caused on my work (if any).

*Ho Chi Minh City, day 11 month 03 year 2025*

*Author*

*Vo Nhat Hao*

*Pham Van Phuc*

# TITLE

# ABSTRACT


- This report explores various learning methods in machine learning based on the Gradient Descent Algorithm (GDA), including its variants such as Batch Gradient Descent (BGD), Stochastic Gradient Descent (SGD), Mini-batch Gradient Descent (MBGD), Momentum, RMSprop, and Adam. These methods are essential for optimizing machine learning models, helping to minimize the loss function and enhance model performance.
    1. **Gradient Descent Algorithm (GDA):** A fundamental optimization algorithm in machine learning, GDA is used to find the minimum value of the loss function. It operates by iteratively updating the model's parameters in the direction of the steepest descent of the loss function.
    2. **Batch Gradient Descent (BGD):** This method computes the gradient over the entire dataset to update the model's weights. While BGD offers high accuracy and stability, it is computationally expensive and time-consuming, especially for large datasets.
    3. **Stochastic Gradient Descent (SGD):** SGD updates the model's weights after processing each randomly selected data sample. It is faster and more memory-efficient than BGD but can introduce noise during the optimization process.
    4. **Mini-batch Gradient Descent (MBGD):** MBGD strikes a balance between BGD and SGD by updating the weights based on small subsets (mini-batches) of the data. This approach combines the stability of BGD with the efficiency of SGD.
    5. **Momentum:** Momentum is an optimization technique that accelerates convergence by incorporating information from previous update steps. It helps reduce oscillations and escape local minima, making the optimization process more efficient.
    6. **RMSprop:** RMSprop is an adaptive learning rate optimization method that adjusts the learning rate based on the squared gradient. It is particularly effective for stabilizing the optimization process in deep neural networks.

7. **Adam:** Adam combines the advantages of Momentum and RMSprop, making it one of the most widely used optimization algorithms in deep learning. It automatically adjusts the learning rate and reduces oscillations during optimization, leading to faster and more stable convergence.

- The report also presents a classification problem using machine learning methods, covering steps from data selection and cleaning to the application of algorithms such as K-Nearest Neighbors (KNN), Logistic Regression, Support Vector Machines (SVM), Decision Trees, and K-Means Clustering. The results are evaluated using metrics such as accuracy, precision, recall, and F1-score.
- Finally, the report discusses the phenomenon of overfitting and methods to mitigate it, including reducing model complexity, applying regularization techniques, using cross-validation, and increasing the size of the training dataset. These strategies help improve the generalization ability of machine learning models, ensuring better performance on unseen data.

# TABLE OF CONTENTS

# LEARNING METHODS IN MACHINE LEARNING ACCORDING TO GDA ALGORITHM (GRADIENT DESCENT ALGORITHM)

## 1. Presenting learning methods in machine learning using the GDA (Gradient Descent Algorithm) algorithm, including: Batch GD, Stochastic GD, Mini-batch GD, Momentum, RMSprop, Adam;

### 1.1. Introduction to Gradient Descent Algorithm

#### a. What is Gradient Descent Algorithm?

- **Gradient Descent Algorithm (GDA):** is an important optimization algorithm in Machine Learning and Mathematical Optimization, used to find the minimum value of a function, optimizing the loss function to determine the optimal set of parameters for models such as Linear Regression or K-means Clustering.
- **Objective:** to minimize prediction errors by adjusting the model's weights in the direction of the steepest decrease of the loss function.
- In general, finding the point with the smallest value of loss functions in Machine Learning is very complex, and sometimes even infeasible.
- Reasons may stem from the complexity of the derivative's form, the high dimensionality of the data points, or the sheer volume of data points.
- Instead, people often try to find local minima and, to some extent, consider these as the solutions to the problem, then use an iterative method to gradually approach the desired point, until the derivative is close to 0.

#### b. Steps to implement Gradient Descent Algorithm

- **Step 1:** Initialize the parameters
- **Step 2:** Calculate the derivative
- **Step 3:** Update the parameters in a decreasing direction
- **Step 4:** Repeat steps 2 and 3 until convergence

#### c. Formula

$$w_j^{(k+1)} = w_j^k - \Delta w_j = w_j^k - \mu \times \frac{\partial Loss}{\partial w_j}$$

#### d. Operating principle

- Gradient Descent operates based on the principle of derivatives. Specifically, the derivative of a function at a point indicates the direction in which the function's value increases most rapidly. Therefore, to find the minimum, we need to move in the opposite direction of the derivative.

**e. Parameter Update Formula**

$$\theta := \theta - \alpha \times \nabla J(\theta)$$

**Where:**

- $\theta$ is the model's parameter (e.g., the weights of a neural network or the coefficients in linear regression).
- α is the learning rate, which determines the size of the step taken.
- $\nabla J(\theta)$ is the gradient of the loss function $J(\theta)$, indicating the direction in which the parameters need to be adjusted to reduce the value of the loss function.
- By updating the parameters based on the gradient, the algorithm gradually brings the model to an optimal value.

**f. Loss function formula**

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(y_i - y_i)^2$$

**Where:**

- $y_i$ is the actual value.
- $y_i$ is the predicted value of the model.
- m is the number of data samples.

General Gradient Descent Algorithm in Multidimensional Space

In the general case, if the loss function depends on many variables:

$$J(\theta_1, \theta_2, ..., \theta_n)$$

Then gradient is a vector containing partial derivatives of J with respect to each variable:

$$\nabla J = \left( \frac{\partial J}{\partial \theta_1}, \frac{\partial J}{\partial \theta_2}, ..., \frac{\partial J}{\partial \theta_n} \right)$$

The update rule becomes:

$$\theta_i := \theta_i - \alpha \frac{\partial J}{\partial \theta_i} \ , \ \forall i \ , \ i = 1, 2, \ \ldots, \ n$$

This process is repeated until convergence.

- Gradient Descent has many variations to apply to many different types of specific problems, including: Batch GD, Stochastic GD, Mini-batch GD, Momentum, RMSprop, Adam.

## 1.2. Batch Gradient Descent (BGD)

### a. What is Batch Gradient Descent (BGD)?

- **Batch Gradient Descent (BGD):** is an optimization method used in machine learning, particularly in training deep learning models. The goal of BGD is to fine-tune the model's weights to minimize the value of the loss function.

### b. How does Batch Gradient Descent work?

- **Batch Gradient Descent** calculates the gradient of the entire dataset to update the weights. At each training step, the gradient of the loss function is computed across all training data points, and the model's weights are updated accordingly.

### c. Formula

$$\theta = \theta - \eta \times \nabla \theta J(\theta)$$

**Where:**

- $\theta$ represents the model's weights.
- $\eta$ is the learning rate.
- $\nabla \theta J(\theta)$ is the gradient of the loss function.

### d. Advantages of BGD

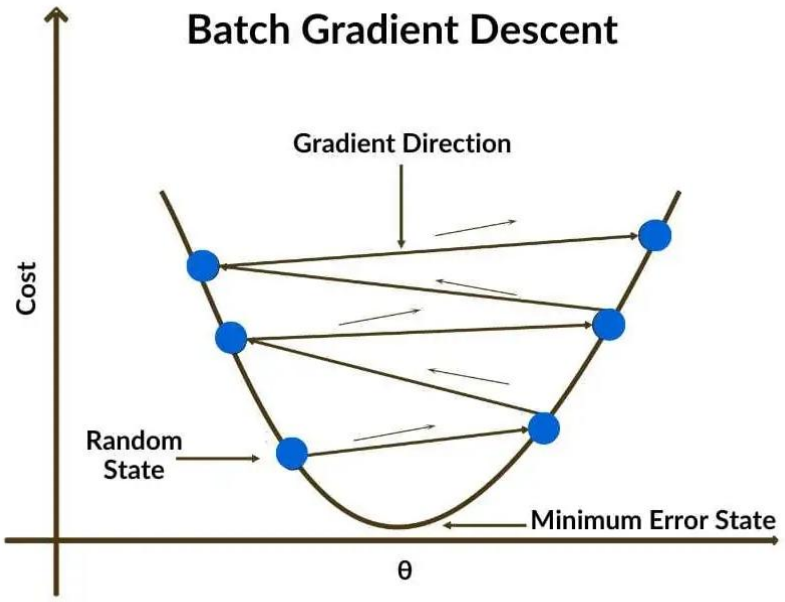
1. **Accuracy and stability:** BGD calculates the gradient over the entire dataset, providing accurate updates that lead to stable convergence.
2. **Stable loss function:** Updates computed from the gradient over the full dataset result in a smoother and more stable loss function.

### e. Disadvantages of BGD

1. **Slow and resource-intensive:** Computing the gradient for the entire dataset at each step can be very time-consuming and memory-intensive.

2. **Not suitable for large datasets:** For massive datasets, BGD is impractical as it requires loading and processing the entire dataset with each update, leading to memory overload.

**f. Illustration**




## 1.3. Stochastic Gradient Descent (SGD)

**a. What is Stochastic Gradient Descent (SGD)?**

- **Stochastic Gradient Descent (SGD):** is a variant of the Gradient Descent algorithm designed to optimize machine learning models efficiently.

**b. How does Stochastic Gradient Descent work?**

- Instead of using the entire dataset, Stochastic Gradient Descent (SGD) calculates the gradient and updates the weights after processing a single randomly selected training example. Each time the model passes through the entire dataset once is called an epoch. Each epoch consists of N updates, where N is the number of data points.
- This process is applied to each individual data point across the entire dataset, then repeated. After each epoch, the order of the data must be shuffled to ensure randomness, helping SGD avoid getting trapped in local minima and improving optimization performance. This very simple algorithm, in practice, works highly effectively.

**c. Formula**

$$\theta = \theta - \eta \nabla \theta J(\theta; x\,(i)\,, y\,(i)\,)$$

**Where:**

- **θ** represents the model's weights.
- **η** is the learning rate.
- $\boldsymbol{x(i), y(i)}$ is a single data point from the training dataset.
- $\boldsymbol{\nabla \theta J(\theta; x(i), y(i))}$ is the gradient of the loss function with respect to θ, calculated for the sample.

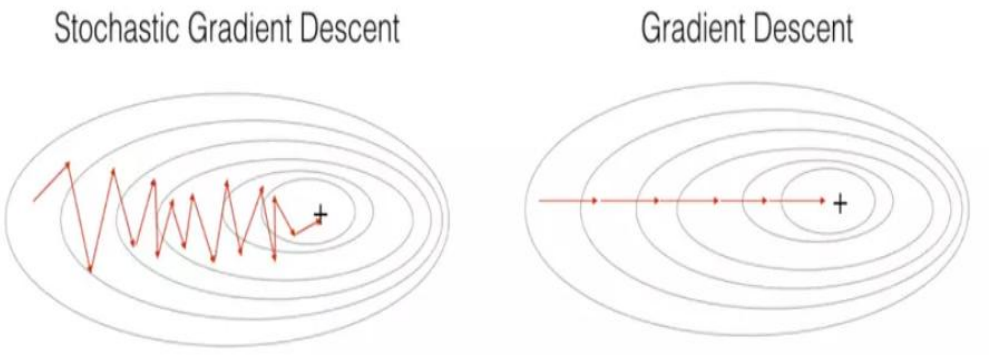
**d. Advantages of SGD**

1. **Faster updates:** Each weight update requires only one data point, making SGD faster than BGD.
2. **Memory-efficient:** Processes only one sample at a time, reducing memory usage.
3. **Escapes local minima:** The randomness in SGD helps it escape local minima and move toward the global minimum.

**e. Disadvantages of SGD**

1. **Noisy updates:** Frequent updates cause fluctuations, making the optimization path noisy.
2. **Slow convergence:** Requires more iterations to converge compared to BGD.
3. **Learning rate sensitivity:** SGD is sensitive to the choice of learning rate, requiring careful tuning.

**f. Illustration**



## 1.4. Mini-batch Gradient Descent (MBGD)

### a. What is Mini-Batch Gradient Descent (MBGD)?

- **Mini-Batch Gradient Descent (MBGD):** is a hybrid optimization method that combines the strengths of BGD and SGD. Instead of processing the entire dataset (BGD) or a single sample (SGD), MBGD updates the weights based on a small subset (mini-batch) of the data.

### b. How does Mini-Batch Gradient Descent (MBGD) work?

- **Mini-Batch Gradient Descent (MBGD)** performs the following steps in each epoch:
  1. Randomly shuffle the entire dataset.
  2. Divide the dataset into mini-batches of size n (the final mini-batch may be smaller if N is not divisible by n).
  3. Iterate through each mini-batch:
     - Compute the gradient on the mini-batch using a small group of randomly selected data points.
     - Update the parameter θ based on this gradient.
  4. Repeat the process until convergence.

### c. Formula:

$$\theta = \theta - \eta \times \frac{1}{m} \sum_{i=1}^{m} \nabla_0 J\left(\theta; x^i, y^i\right)$$

**Where:**

- $\theta$ represents the model's weights.
- $\eta$ is the learning rate.
- $m$ is the mini-batch size.
- $\nabla_0 J\left(\theta; x^i, y^i\right)$ is the gradient of the loss function for the i-th sample in the mini-batch.

### d. Advantages of MBGD

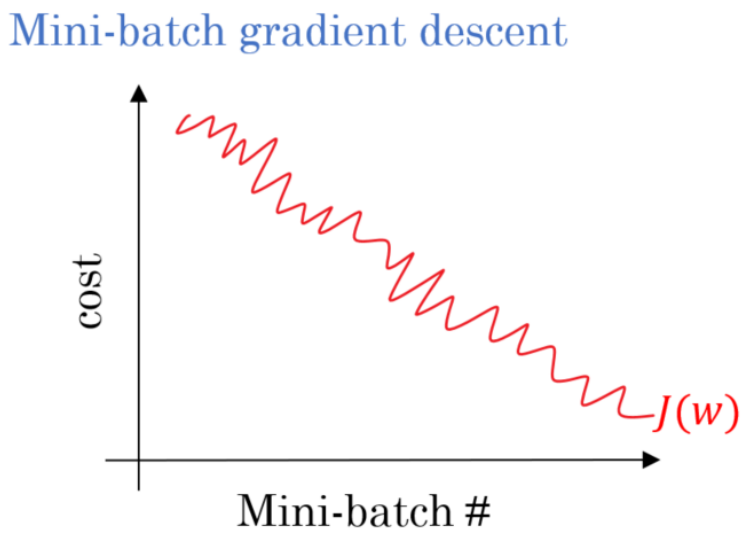
1. **Efficient computation:** Faster than BGD because it only processes a small batch.
2. **Balance between speed and stability:** Provides smoother updates than SGD while being more efficient than BGD.
3. **Lower memory requirement:** Processes small batches instead of the

entire dataset, making it suitable for large datasets.

**e. Disadvantages of MBGD**

1. **Hyperparameter tuning:** The size of the mini-batch affects performance and requires tuning.
2. **Learning rate adjustment:** The learning rate needs careful adjustment for optimal performance.

**f. Illustration**




## 1.5. Momentum

**a. What is Momentum?**

- **Momentum** is an optimization technique used in algorithms like Gradient Descent (GD) to improve convergence speed and escape local minima. Momentum accelerates the optimization process by leveraging gradients from previous steps to adjust the update velocity in subsequent steps. This method draws inspiration from physics: if an object is in motion, it continues moving in the direction of its velocity (momentum) rather than abruptly changing direction based on the current gradient. It helps avoid getting trapped in local minima and enhances convergence speed.

**b. How does Momentum work?**

- Gradient Descent updates the parameters (weights and biases) to minimize the loss function. However, in some cases, traditional Gradient Descent can be slow or get stuck in local minima. Momentum is introduced to address this issue. When the gradient consistently points in the same direction, it gets

accelerated. When the gradient oscillates between two directions, Momentum reduces the oscillation to converge faster.

- The idea is to add a portion of the previous update (from earlier iterations) to the current update, making the optimization process faster and more stable.

**c. Formula:**

- The formula for Gradient Descent with Momentum is an extension of the standard Gradient Descent update rule. This formula incorporates the velocity vvv, which represents the momentum of the optimization process:

$$v_t = \beta v_{t-1} - \eta \nabla_\theta J(\theta)$$
$$\theta = \theta + v_t$$

**Where:**

- $v_t$ is the velocity vector at time t (momentum speed).
- β is the momentum parameter, usually set between 0 and 1. It controls the impact of momentum in the update process.
- η is the learning rate, determining the size of the weight update step.
- $\nabla_\theta J(\theta)$ is the gradient of the loss function with respect to the parameters θ (weights and biases).
- $\theta$ is the model's parameters (weights and biases).

**Meaning:**

- $v_t$ represents momentum, combining both the current gradient and the previous gradients.
- When the gradients gradually change, momentum helps maintain the direction of optimization, allowing the parameters to move faster toward the minimum.
- If the gradients change drastically, momentum helps reduce oscillations and maintains algorithm stability.

**d. Advantages of Momentum**

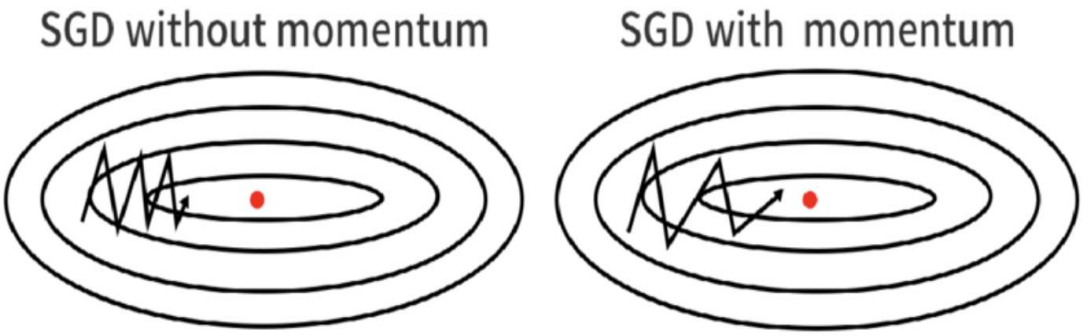
1. **Faster Convergence**: Momentum accelerates the optimization process by using momentum from previous updates to push the algorithm faster toward the minimum.

2. **Reduced Oscillation**: When encountering regions with weak gradients or oscillations, momentum helps reduce the fluctuations during optimization, making the algorithm more stable and preventing it from getting stuck in local minima.
3. **Ability to Escape Local Minima**: Momentum helps the algorithm escape local minima by maintaining a consistent direction for updates, rather than getting trapped in insignificant local minima.
4. **Improved Accuracy**: By maintaining momentum, the algorithm not only converges faster but also achieves better accuracy in complex deep learning tasks.

**e. Disadvantages of Momentum**

1. **Requires Tuning Parameters**: The momentum parameter $\beta$ and learning rate $\eta$ need to be chosen carefully; otherwise, they can cause the algorithm to converge too quickly or too slowly.
2. **Difficulty Handling Sparse Data**: Momentum may not be effective when dealing with sparse data or unclear features.
3. **Ineffective for Non-Uniform Loss Functions**: Momentum may not adjust the parameters correctly in problems where the loss function changes non-uniformly.
4. **Risk of Escaping Global Minimum**: If $\beta$ is not tuned properly, momentum can overshoot the global minimum without returning.
5. **Heavy Oscillations**: If not properly adjusted, momentum can lead to excessive oscillations during the optimization process.

**f. Illustration**




## 1.6. RMSprop

**a. What is RMSProp?**

- RMSProp (Root Mean Square Propagation) is an adaptive learning rate

optimization method designed to address issues encountered during the training of neural networks, especially in deep learning tasks. RMSProp is an extension of the Adagrad algorithm and helps reduce computational effort while improving the stability of the optimization process.

**b. How does RMSProp work?**

- RMSProp adjusts the learning rate for each parameter based on the squared gradients of the parameters from previous iterations. Specifically, RMSProp reduces the learning rate when the gradients are small, allowing for faster and more stable convergence.

**c. Formula:**

- Accumulated gradient:

$$E_t = \gamma E_{t-1} + (1-\gamma) g_t^2$$

- Parameter update:

$$\theta_{t+1} = \theta_t - \alpha \frac{g_t}{\sqrt{E_t + \varepsilon}}$$

- Initialization:
  - Learning rate: $\alpha$
  - Decay rate: $\gamma$
  - Small constant for numerical stability: $\varepsilon$
  - Initial parameter values: $\theta$
- Where:
  - $g_t$ is the gradient of the loss function with respect to the parameters at time t.
  - $\gamma$ is the decay factor controlling the influence of past gradients.
  - $E_t$ is the moving average of squared gradients.
  - $\alpha$ is the learning rate.
  - $\varepsilon$ is a small constant to avoid division by zero.

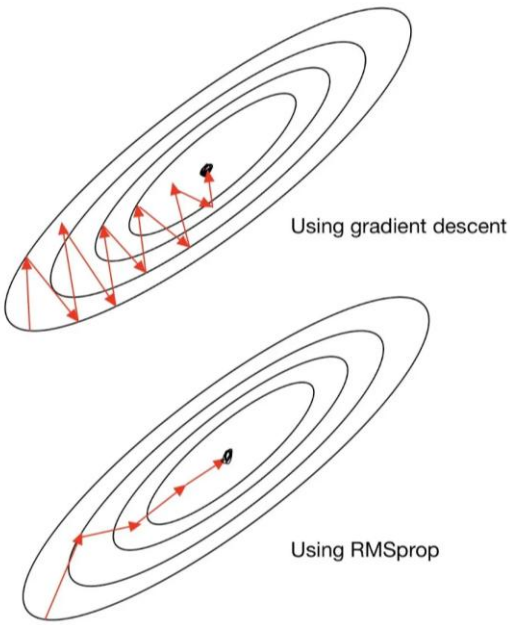
**d. Advantages of RMSProp**

1. **Automatic Learning Rate Adjustment**: RMSProp adjusts the learning rate for each parameter based on historical gradients, improving the stability of the training process.
2. **Effective for Deep Networks**: It works well for deep neural networks, especially when the data is uneven or the gradients have large variations.
3. **Stabilizes Gradient**: By reducing the learning rate when the gradient is small, it stabilizes the optimization process.

**e. Disadvantages of RMSProp**

1. **Dependence on Parameters**: γ\gammaγ and α\alphaα need to be carefully tuned, which may require experimentation.
2. **Risk of Early Convergence**: The gradually decreasing learning rate can lead to early convergence at a suboptimal minimum.

**f. Illustration**




## 1.7. Adam

**a. What is Adam?**

- **Adam (Adaptive Moment Estimation)** is an efficient and widely used optimization algorithm in machine learning, especially for training deep learning models with large datasets. Adam combines the benefits of two previous optimization methods: Momentum and RMSProp.

**b. How does Adam work?**

- Adam combines **gradient descent** with **momentum** and **RMSProp** to update the weights during training, helping to accelerate convergence and reduce oscillation.

**c. Method**

- **Momentum**: Adam uses the exponentially weighted average (momentum) of the gradients, helping the algorithm move faster toward the minimum and reducing oscillation.
- **Momentum update formula:**

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\frac{\partial L}{\partial \mathrm{w}_t}$$

**Where:**

- $m_t$ is the momentum at time step t.
- $\beta_1$ is the momentum parameter, typically chosen as 0.9.
- $\frac{\partial L}{\partial \mathrm{w}_t}$ is the gradient of the loss function with respect to the weights at time t.

- **RMSProp**: Adam also uses the exponentially weighted average for the square of the gradient (similar to RMSProp), which adjusts the learning rate for each parameter based on the gradient history.
- **RMSProp update formula:**

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)\left(\frac{\partial L}{\partial \mathrm{w}_t}\right)^2$$

**Where:**

- $v_t$ is the velocity (squared gradient) of the gradient at time step t.
- $\beta_2$ is the parameter controlling the rate of decay for the squared gradient (typically 0.999).

**d. Formula**

- **Parameter Update:** Adam uses information from both momentum and RMSProp to adjust model parameters:

$$\theta_{t+1} = \theta_t - \alpha \frac{m_t}{\sqrt{v_t + \varepsilon}}$$

**Where:**

- $\theta_{t+1}$ is the weight value after the update.
- $\alpha$ is the learning rate.
- $\varepsilon$ is a small constant (typically $10^{-8}$) to avoid division by zero.

- **Bias correction:** Since the values of mtm_tmt and vtv_tvt are initialized to 0, they tend to be biased towards 0 at the beginning of the training process. To correct this bias, Adam computes the bias-corrected parameters.

$$m_t = \frac{m_t}{1-\beta_1^t}, \qquad v_t = \frac{v_t}{1-\beta_1^t} m$$

- **Final updated formula (with bias correction):**

$$\theta_{t+1} = \theta_t - \alpha \frac{m_t}{\sqrt{v_t + \varepsilon}}$$

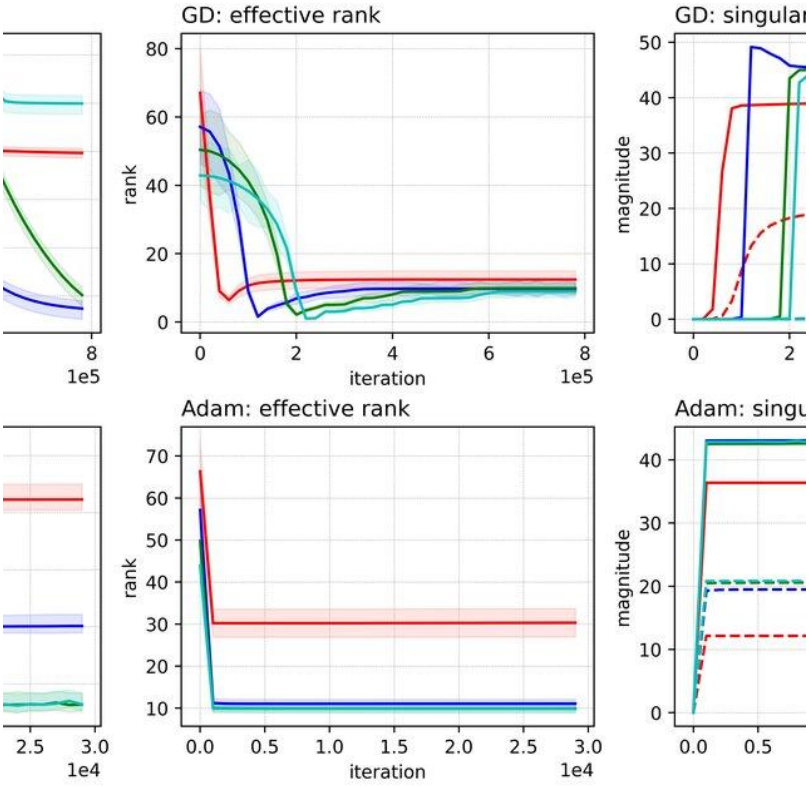
**e. Advantages of Adam**

1. **Automatic Learning Rate Adjustment**: Adam adjusts the learning rate for each parameter, improving optimization speed and stability.
2. **Faster Convergence**: With the combination of momentum and RMSProp, Adam optimizes models quickly, especially for large deep learning models.
3. **Reduces Oscillation**: Adam helps reduce oscillation during optimization, especially in problems with unstable gradients.
4. **Low Memory Usage**: Compared to other optimization methods, Adam requires less memory and computation.

**f. Disadvantages of Adam**

1. **Dependency on Hyperparameters**: The choice of α, $\beta_1$, and $\beta_2$ requires experimentation to achieve optimal performance.
2. **Adjustment in Special Scenarios**: While Adam is powerful, it may not work as well for extremely complex models or in highly sensitive tasks.

**g. Illustration**





## 2. Examples of algorithms

- Using two types of data to evaluate the performance of different optimization algorithms, including Gradient Descent (Batch Gradient Descent, Stochastic

Gradient Descent, Mini-batch Gradient Descent, Momentum, RMSprop, and Adam). Specifically, we will conduct experiments with:

- Random Data: This data is generated randomly to test the ability of algorithms to learn from simple data samples, with no complex relationships between features and the target variable. The goal is to observe how machine learning algorithms handle randomly structured data, and then compare the effectiveness of each algorithm in optimizing the objective function.
- Real Data: After testing with random data, we proceed with real-world data from a regression problem. Real data is more complex and contains less clearly defined factors compared to random data, creating a more realistic environment for evaluating the accuracy and generalization capabilities of the algorithms.

- **Example with random data**

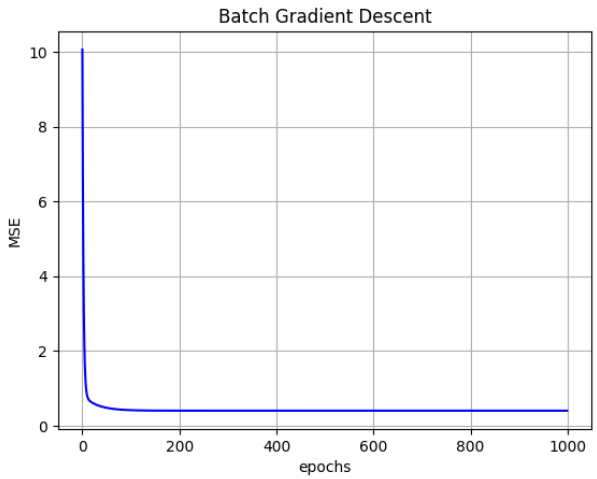
Random data helps determine the effectiveness of each algorithm in an environment free from noise and uncertain factors. The algorithms will be tested on simple datasets with a small number of samples and randomly generated features. The goal is to assess the algorithms' ability to converge quickly and accurately under ideal conditions. This allows us to observe the differences in the convergence behavior of each algorithm.

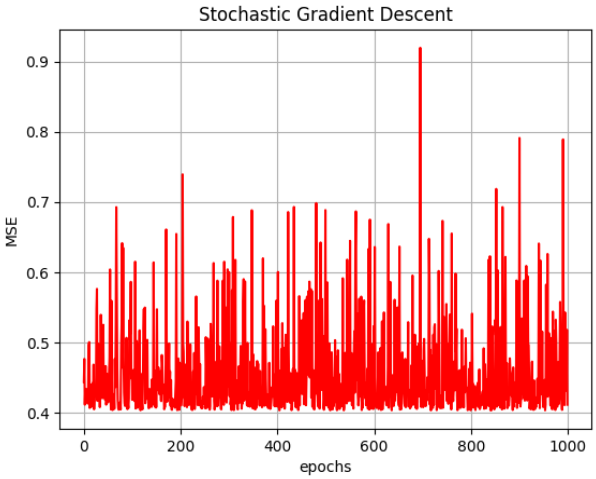
**Data description:**

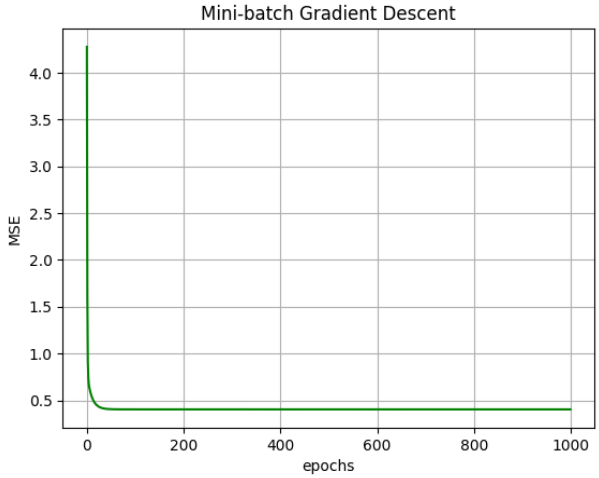
```
# Create random data
np.random.seed(42)
X = 2 * np.random.rand(100, 1)  # Random data (100 samples, 1 feature)
y = 4 + 3 * X + np.random.randn(100, 1)  # Linear relationship with noise
```

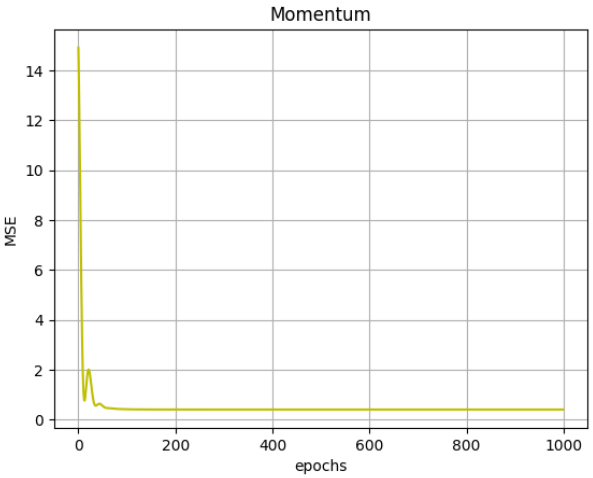
- The data X consists of 100 randomly generated samples within the range [0, 2), with each sample having a single feature value.
- y has a linear relationship with X: the value of y increases as X increases, with a bias term of 4 and a weight of 3.
- Noise is added to make the relationship between X and y not perfectly accurate, resembling real-world data that may contain errors or unmeasured factors.

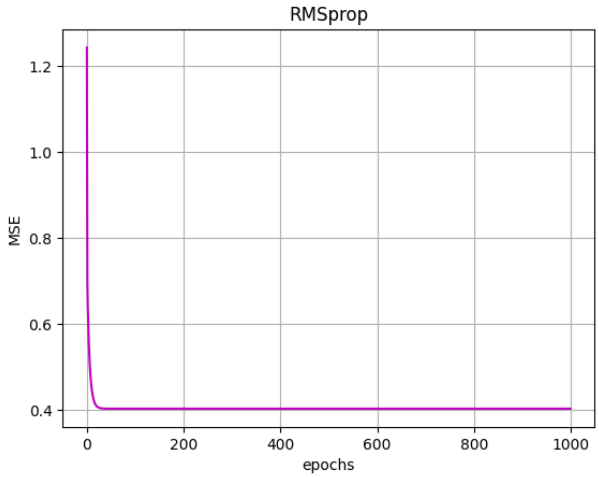
- **Result:**

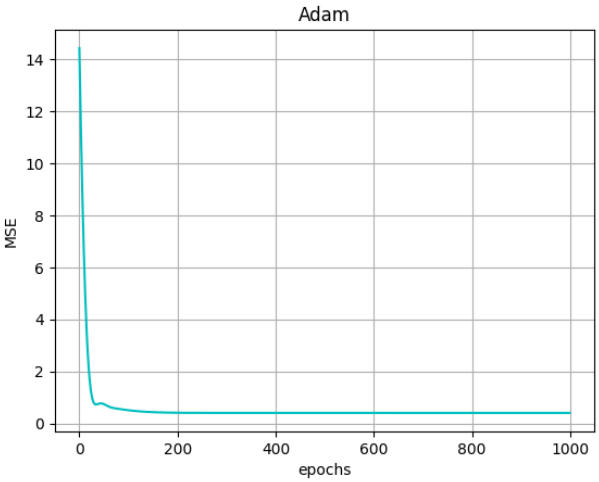

Batch Gradient Descent
MSE
epochs

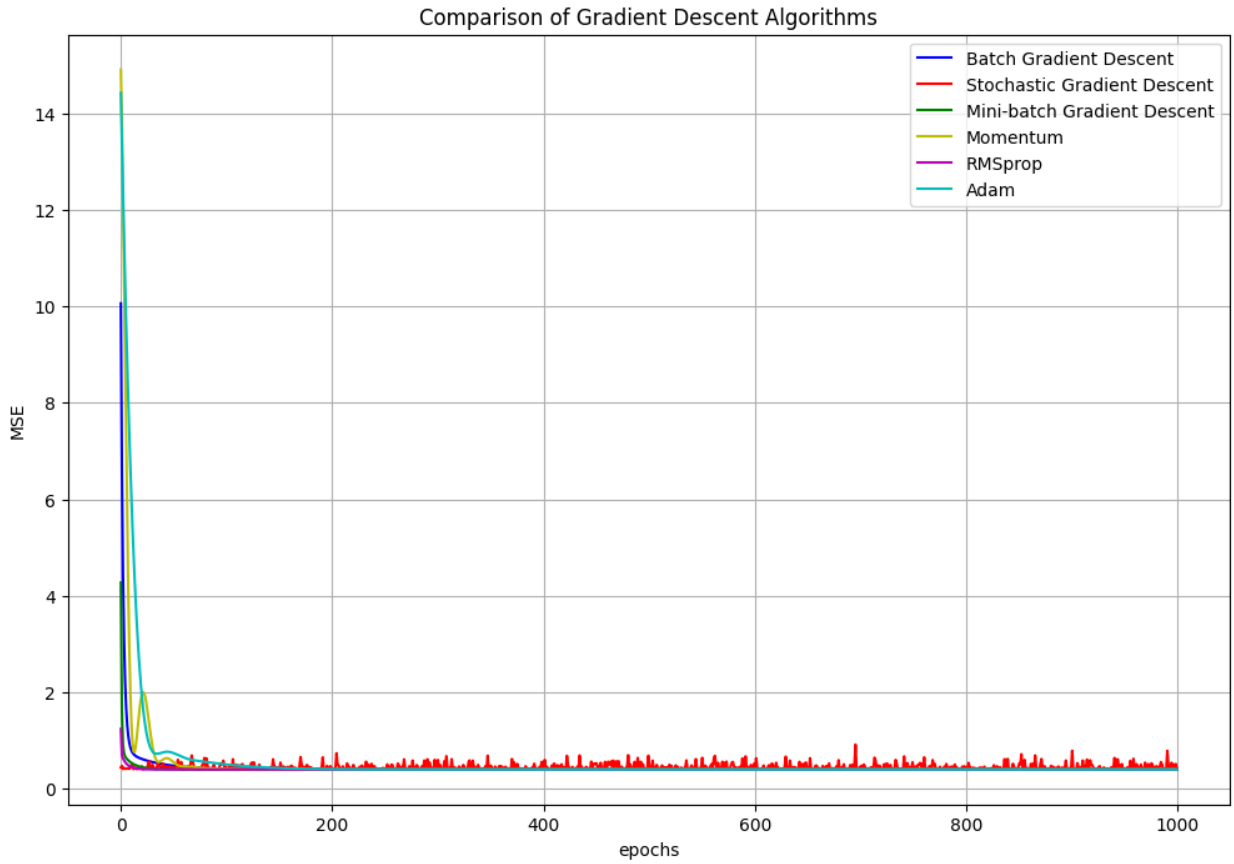

Stochastic Gradient Descent
MSE
epochs


Mini-batch Gradient Descent
MSE
epochs


Momentum
MSE
epochs


RMSprop
MSE
epochs


Adam
MSE
epochs



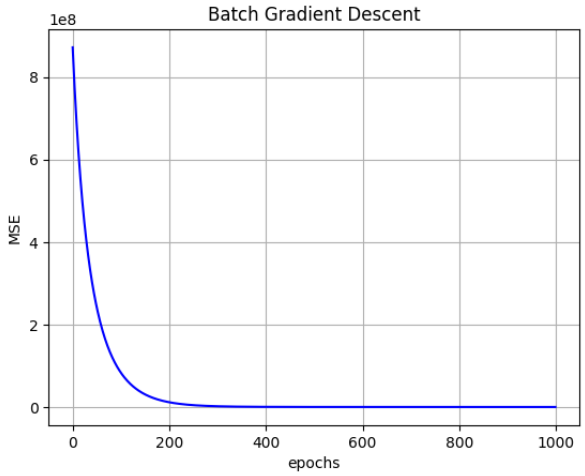

- **Example with Wholesale customers data from UCI**

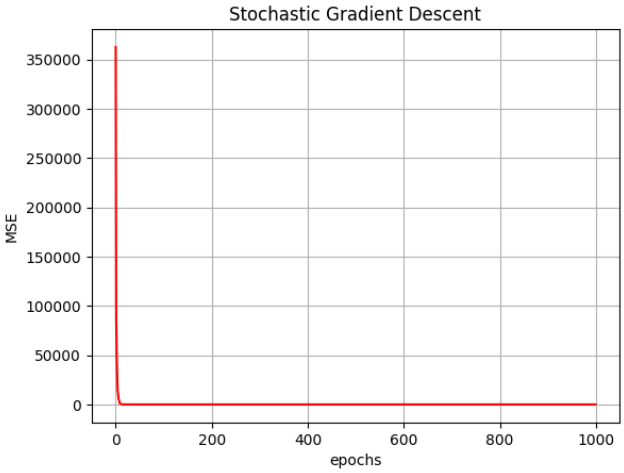
  Real data reflects the complex real-world problems that models face, such as the existence of noise in the data, non-linear relationships, or correlated features. Using real-world data helps test the algorithms' ability to learn and predict accurately when dealing with real-world conditions.

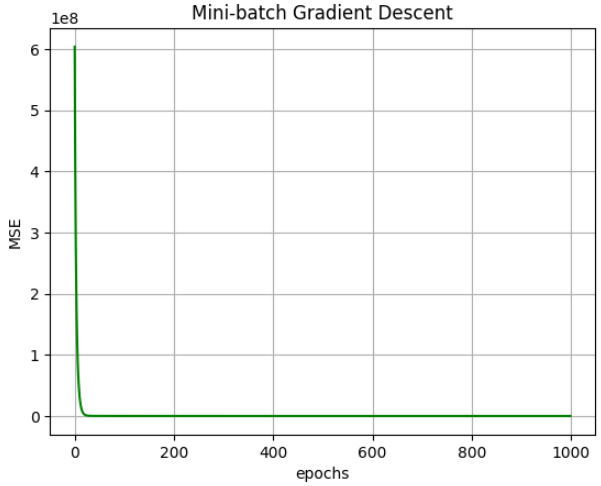
  **Data description:**

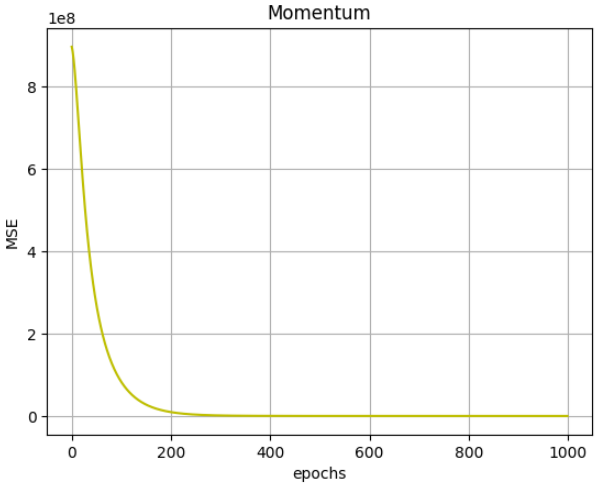
  The Wholesale Customer Data is a dataset provided by the UCI Machine Learning Repository, containing 7 features and 440 instances. It describes information about wholesale customers in the food and household product retail industry. This dataset includes details related to the consumption of products by wholesale customers in stores, helping model trends and consumer behaviors.

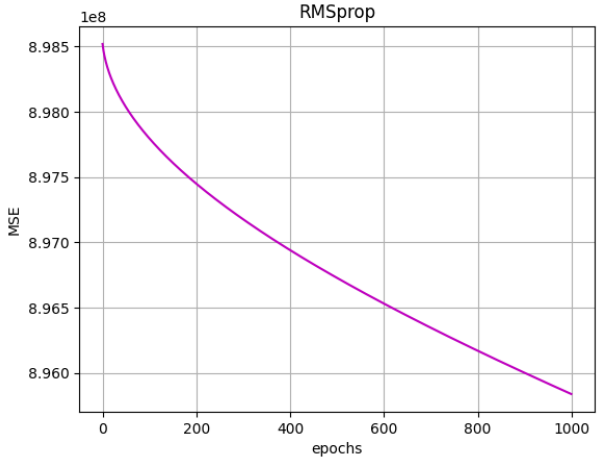
- **Result:**

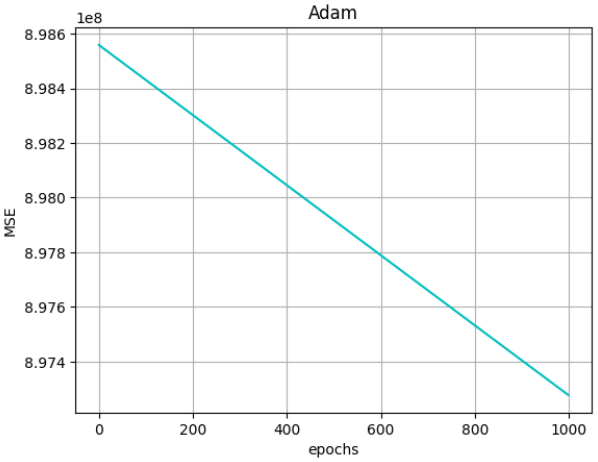

Batch Gradient Descent
1e8
MSE
epochs

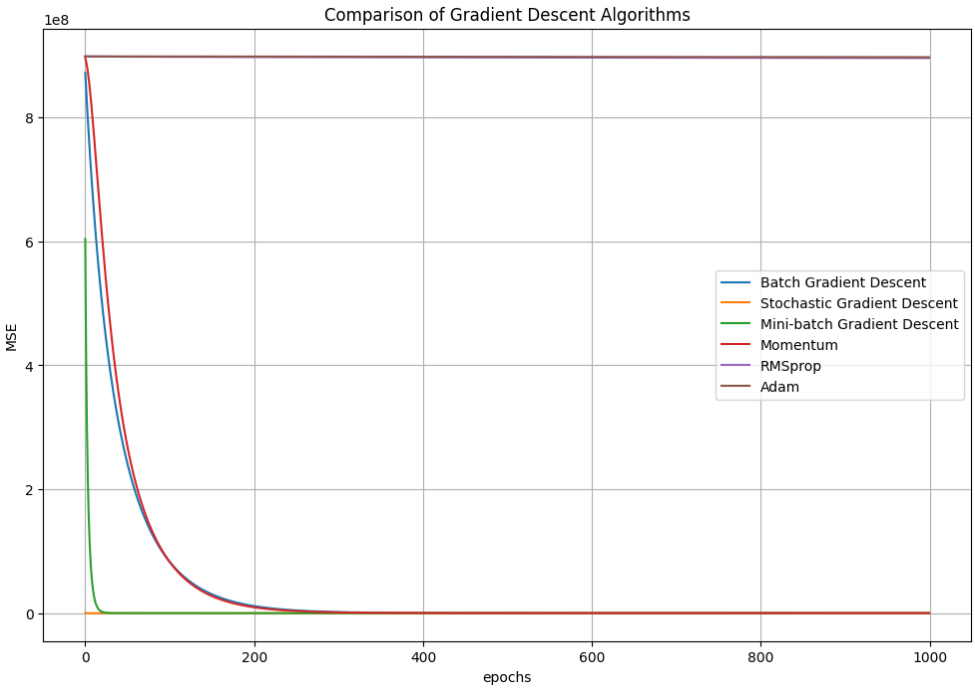

Stochastic Gradient Descent
MSE
epochs


Mini-batch Gradient Descent
1e8
MSE
epochs


Momentum
1e8
MSE
epochs


RMSprop
1e8
MSE
epochs


Adam
1e8
MSE
epochs




## 3. Application to the problem of predicting house prices

- Problem Description: The Housing problem from the UCI Machine Learning Repository is a regression problem where the goal is to predict the value (price) of houses based on their features. This dataset helps analyze and predict the price of a house based on various factors such as area, number of bedrooms, number of bathrooms, and other amenities. The dataset contains 13 features and 545 instances, which represent different attributes of the houses and their respective prices.
- **Result:**

```
Detailed results for each algorithm:

[Batch GD]:
  Estimated house price: 7173661.551936084
  MSE value: 0.0077695806810960015
--------------------------------------

[SGD]:
  Estimated house price: 8039252.42586353
  MSE value: 0.006540739516230903
--------------------------------------

[Mini-batch GD]:
  Estimated house price: 7937664.973152925
  MSE value: 0.006401910723985806
--------------------------------------

[Momentum]:
  Estimated house price: 7175833.663559767
  MSE value: 0.007769195688054719
--------------------------------------

[RMSprop]:
  Estimated house price: 7616073.7905556355
  MSE value: 0.00713475625395747
--------------------------------------

[Adam]:
  Estimated house price: 7943124.826047742
  MSE value: 0.006415465980413443 5
--------------------------------------
```

# SOLVING A CLASSIFICATION PROBLEM USING MACHINE LEARNING METHODS

## 1. Select a data set (on UCI) with diverse features including both numeric and categorical; data has not been used in class exercises

- The selected data is Online Shoppers Purchasing Intention Dataset data taken from UCI site
- Data link: Online Shoppers Purchasing Intention Dataset - UCI Machine Learning Repository
- Data description: This dataset contains information about users' browsing behavior on an e-commerce site. It includes 12,330 sessions, with 18 features divided into:
    - 10 Numerical Features
    - 8 Categorical Features
- Objective: Predict whether a user will make an online purchase based on browsing behavior.
- Data details:

| Attribute Name | Data Type | Description |
|---|---|---|
| Administrative | Numerical | Number of visits to administrative pages in the session |
| Administrative_Duration | Numerical | Total time (seconds) spent on administrative pages |
| Informational | Numerical | Number of visits to informational pages |
| Informational_Duration | Numerical | Total time (seconds) spent on informational pages |
| ProductRelated | Numerical | Number of visits to product-related pages |
| ProductRelated_Duration | Numerical | Total time (seconds) spent on product pages |
| BounceRates | Numerical | Bounce rate of the session |
| ExitRates | Numerical | Exit rate of the last page in the session |
| PageValues | Numerical | Average value of the page visited before conversion |
| SpecialDay | Numerical | Value from 0 to 1, representing the session's relevance to special days (e.g., Black Friday, |

| | | |
|---|---|---|
| | | Christmas, etc.) |
| Month | Categorical | Month in which the session occurred (month name) |
| OperatingSystems | Categorical | Operating system used by the user |
| Browser | Categorical | Type of web browser used |
| Region | Categorical | Geographical region of the user |
| TrafficType | Categorical | Type of traffic source (1 to 20) |
| VisitorType | Categorical | Type of visitor (New Visitor, Returning Visitor, Other) |
| Weekend | Categorical | Whether the session occurred on a weekend (TRUE/FALSE) |
| Revenue | Categorical | Prediction target: Did the user complete a purchase? (TRUE/FALSE) |

## 2. Perform data cleaning and visualization steps to better understand the data

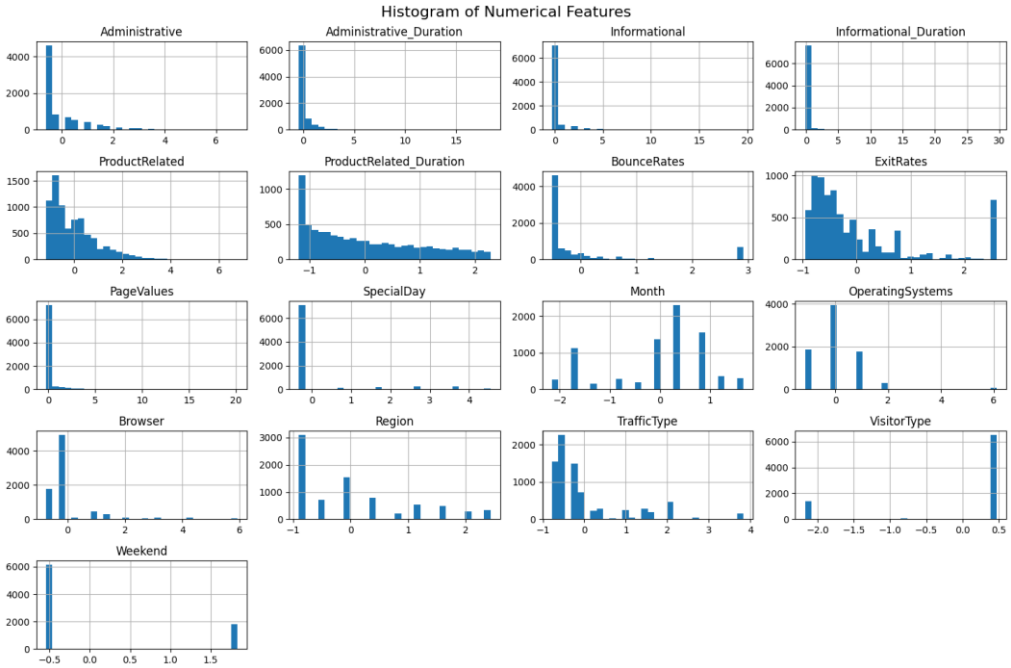
### 2.1. Data cleaning

- Data cleaning is the process of detecting, correcting or removing errors, incorrect, inaccurate or inappropriate data in a data set to ensure data consistency and quality.
- Including: Handling missing data, duplicate data, invalid data, removing some unimportant columns (Operating Systems, Browser).
- Tools used: Libraries and functions of Python language such as:
- Steps to perform:
    - Check data: check data size (number of rows, number of columns), check data type of each column, check for missing data (null, NaN)
    - Handle missing data: If there is little missing data, remove rows/columns, if there is a lot missing, fill in replacement values (mean, median, mode)
    - Handle duplicate data

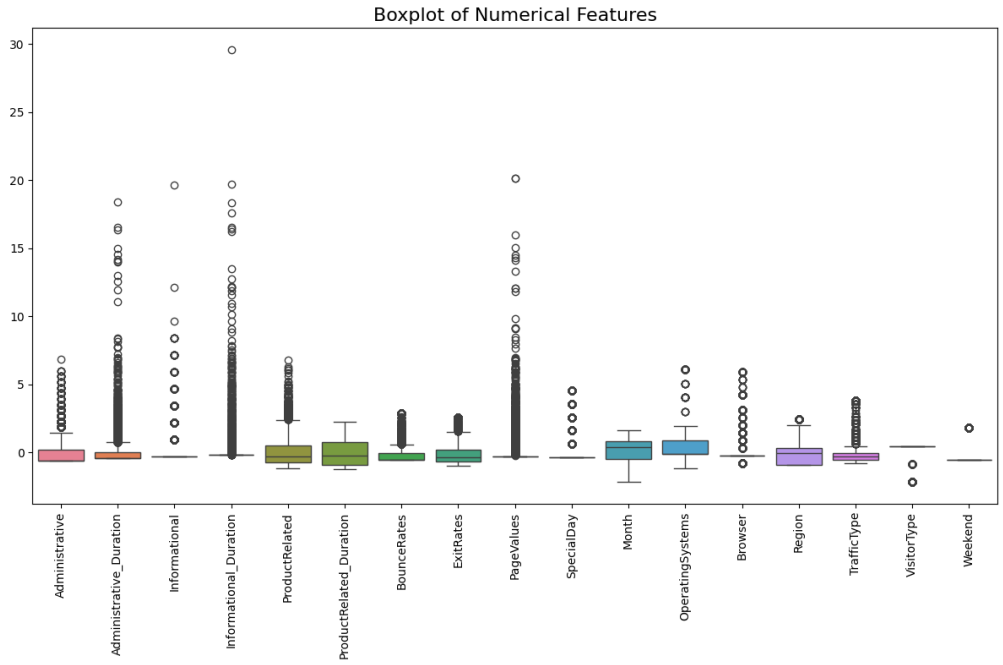
### 2.2. Data Visualization

- Goal: Better understanding of trends, distribution of data
- Includes:

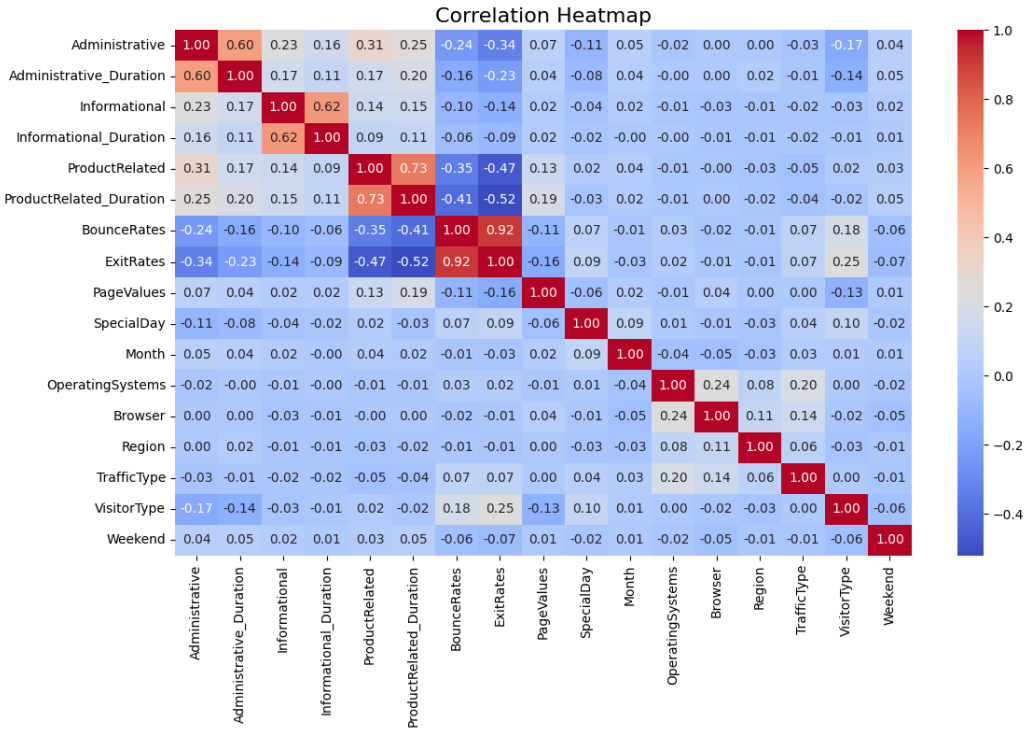
- Drawing histograms to see data distribution:
  - Histogram is used to show the frequency distribution of values in numerical data (continuous characteristics). From here, the distribution of the characteristics can be determined (normal distribution, right-skewed distribution or left-skewed distribution). Extremely large or extremely small values can be easily detected. This chart also shows the dispersion of the data.
  - Results:



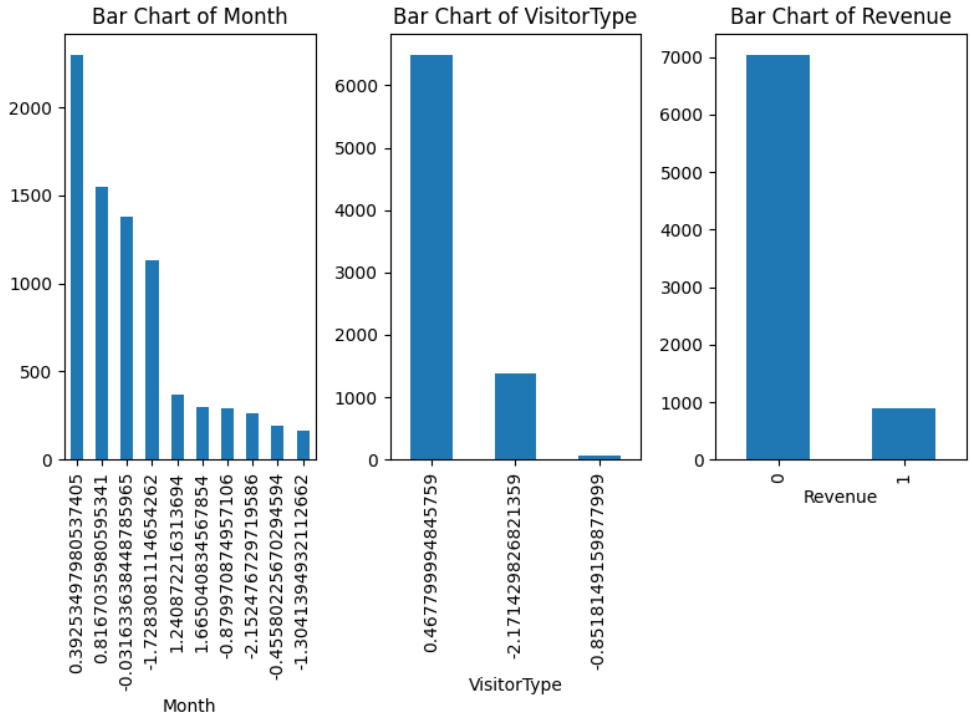

- Drawing boxplots to detect outliers
  - Boxplots help visualize the distribution of data by metrics such as medians, quartiles, and outliers. They clearly show the median, first quartile (Q1), third quartile (Q3), and outliers that fall outside these ranges. Points outside the whisker range of the boxplot are outliers, allowing you to easily identify unusual values in the data. Helps you see whether the data is evenly distributed.
  - Results:




- Drawing heatmaps to examine relationships between variables
  - A heatmap shows the relationships between features in the data as a colored matrix. It helps identify correlations between pairs of variables. Values close to 1 or -1 indicate a strong relationship (positive or negative), while values close to 0 indicate a weak or no relationship. Consider removing some variables to avoid multicollinearity in the machine learning model.
  - Results:



- Bar charts for categorical variables
    - Column charts help you visualize the frequency of values in categorical data. This chart helps you see how often each value appears in categorical variables such as Month, VisitorType, Revenue. It can help you see whether the categories are evenly distributed or concentrated in a few groups.
    - Results:




## 3. Perform data conversion and normalization steps; divide into Train and Test sets

### 3.1. Convert data

- Convert data from text to numeric format for training convenience
- **Month:** Use Label Encoding (Jan = 0, Feb = 1, …) or One-Hot Encoding if you need to keep the distinction between months.
- **OperatingSystems:** Use Label Encoding (Windows = 0, MacOS = 1, …) or One-Hot Encoding if the number of categories is small.
- **Browser:** Use One-Hot Encoding if the number of browser types is small or Binary Encoding if there are too many categories.
- **Region:** Use Label Encoding if the number of regions is large, or One-Hot Encoding if the number of regions is small.

- **TrafficType:** Use Label Encoding if the number of categories is small, or Binary Encoding to reduce the number of dimensions if the categories are large.
- **VisitorType:** Use Label Encoding (New = 0, Returning = 1, Other = 2).
- **Weekend:** Convert to binary (FALSE = 0, TRUE = 1).
- **Revenue:** Convert to binary (FALSE = 0, TRUE = 1).

### 3.2. Normalize data

- Normalize data sets to a common standard. This reduces the variance between features with different units of measurement, thereby improving performance.
- Input of normalizing data: Numeric data set containing values at different scales
- Output of normalizing data: Data that has been normalized to the same range or normal distribution
- Main library for normalizing data in Python: *sklearn.preprocessing with MinMaxScaler() and StandardScaler() functions*

### 3.3. Split into train set and test set

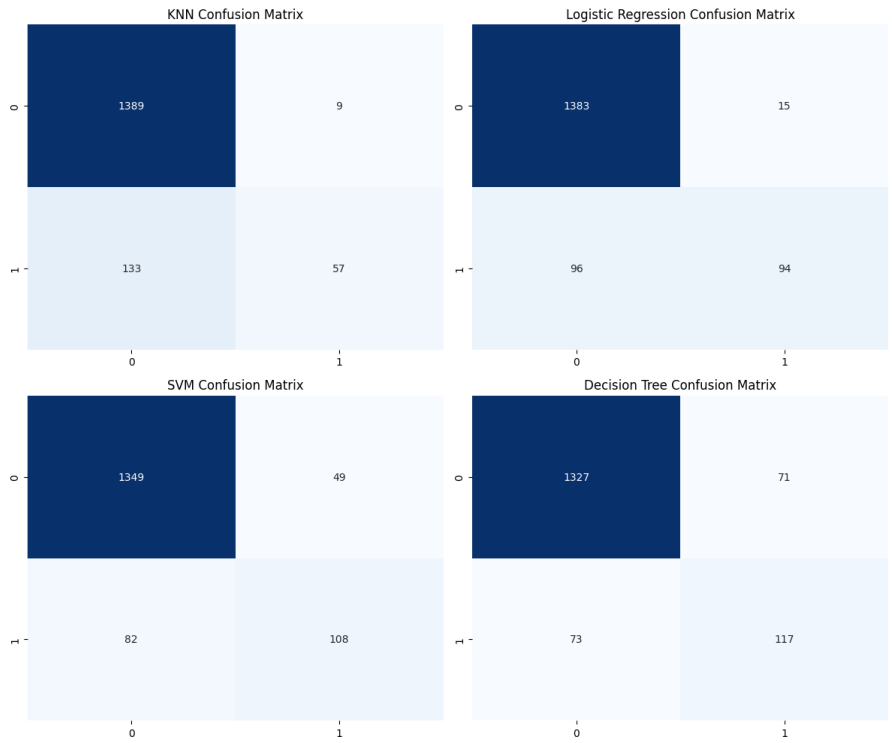
- Once the data has been cleaned and normalized, the next step is to split the data into two parts:
    - **Training Set (80%):** Used to train the model, helping the model learn patterns from the data.
    - **Test Set (20%):** Used to evaluate the accuracy of the model after it has been trained.

## 4. Perform different machine learning algorithms (choose at least 5 algorithms)

- KNN
- Logistic Regression
- SVM
- K-Means clustering
- Decision Tree

## 5. Evaluate results through Test sets; compare methods

- Confusion Matrix: helps to evaluate the performance of a classification model by comparing the predicted values with the actual values. Confusion Matrix shows how well the model has classified the data points correctly or incorrectly.




## 5.1. KNN (K-Nearest Neighbors):

- Accuracy: 91.06% — The accuracy of KNN is quite high.
- Precision: 0.86 — The model is quite good at predicting points belonging to class "1".
- Recall: 0.30 — Despite the high accuracy, the low Recall shows that the model lacks the ability to recognize actual "1" points (i.e., the ability to detect positive classes is still low).
- F1-Score: 0.45 — The F1-Score is quite low, showing that the balance between Precision and Recall is not good.

## 5.2. Logistic Regression:

- Accuracy: 93.01% — Logistic Regression shows very good accuracy, higher than KNN.
- Precision: 0.86 — This model has a higher prediction accuracy, especially for class “1”.
- Recall: 0.49 — Recall is better than KNN, but still quite low, indicating that this model is not really strong in detecting all positive cases.
- F1-Score: 0.63 — Logistic Regression’s F1-Score is quite higher than KNN, indicating that the model has a better balance between Precision and Recall.

## 5.3. SVM (Support Vector Machine):

- Accuracy: 91.75% — The SVM model has the highest accuracy among the models.

- Precision: 0.69 — The Precision is quite high, showing that the SVM model has a relatively good prediction accuracy.
- Recall: 0.57 — The Recall is higher than the previous models, showing that the SVM model has a better ability to identify positive cases.
- F1-Score: 0.62 — The highest F1-Score, showing a good balance between Precision and Recall.

### 5.4. Decision Tree:

- Accuracy: 90.93% — Lower accuracy than Logistic Regression and SVM, but still acceptable.
- Precision: 0.62 — Low precision, which indicates that the model does not predict class "1" effectively.
- Recall: 0.62 — Recall is quite high, indicating that the model can detect a large number of positive cases.
- F1-Score: 0.62 — F1-Score is relatively good, but not as strong as Logistic Regression and SVM.

### 5.5. K-Means Clustering:

- Inertia: 96113.28 — The Inertia is quite large, indicating that the clustering groups are still far apart and the model has not yet clustered effectively. However, for K-Means, Inertia is just a measure of how close points are in the same group. This model cannot be directly compared in accuracy with other classification models.

## 6. Learn about Overfitting and methods to solve overfitting; apply in step 4); compare using methods to avoid overfitting and not using them

### 6.1. Learn about overfitting

#### 6.1.1. Definition of Overfitting

- Overfitting is not an algorithm in Machine Learning. It is an undesirable phenomenon that commonly occurs.
- Overfitting happens when a model becomes too complex and memorizes too many details, including noise and outliers in the training dataset. This means the machine learning model learns the training data too well, to the extent that it fails to generalize properly to new data.
- This can lead to:
    - High variance: The model reacts strongly to small changes in input data, causing instability.
    - Low bias but high error on test data: The model fits the training set very closely (low bias) but generalizes poorly, leading to high errors on test data.
- Example: Consider a machine learning model trained to analyze images and identify pictures of dogs. If the training dataset mainly consists of pictures of dogs in a park, the model might learn to use grass as a distinguishing feature. As a result, it may fail to recognize a dog in an indoor setting.

### 6.1.2. Causes of Overfitting

- **Overly Complex Models**
    - Complex models (such as deep neural networks or high-degree polynomial regression) have a large number of parameters, making them highly flexible in learning complex patterns.
    - However, this also increases the risk of learning noise in the data instead of just meaningful patterns.
- **Insufficient or Noisy Data**
    - When the training dataset is too small, the model tends to learn from a limited number of examples, causing it to treat noise as important patterns.
    - Data with many outliers or inconsistencies can also mislead the model into learning incorrect relationships.
- **Excessive Training (Too Many Epochs in Iterative Algorithms)**

- Training for too long can cause the model to memorize not only the general rules but also the noise in the data, especially in deep learning.

- **Lack of Regularization Mechanisms**
  - Regularization techniques help limit model complexity.
  - Without applying them, the model can freely learn excessive details from the training set, increasing the risk of overfitting.

### 6.1.3. Signs of Overfitting

- **High Accuracy on the Training Set but Low Accuracy on the Test Set**
  - The model performs exceptionally well on the training data but struggles with new, unseen data.
  - This indicates that the model has memorized the training set rather than learning general patterns.
- **High Variance in Predictions**
  - The model produces significantly different outputs with small changes in input data.
  - This happens because it has learned excessive details from the training set, including noise, making it unstable when encountering new data.

### 6.1.4. Consequences of Overfitting

- Overfitting can significantly reduce the performance of a model in practice, with effects such as:
  - **Poor generalization:** The model does not perform well with new data.
  - **Unnecessary complexity:** Makes the model difficult to interpret, maintain, and deploy.
  - **High computational cost:** More complex models require more computational resources.

### 6.1.5. Methods to deal with Overfitting

- **Reduce Model Complexity**

- Choose a simpler model (e.g., linear regression instead of high-degree polynomial regression, or limiting the depth of decision trees).

- **Regularization**
  - **L1 (Lasso) Regularization:** Adds a penalty proportional to the absolute value of coefficients, helping create a model with fewer parameters by eliminating some features.
  - **L2 (Ridge) Regularization:** Adds a penalty proportional to the squared value of coefficients, reducing parameter magnitudes without completely removing them.
  - **Elastic Net:** Combines both L1 and L2 regularization, useful when the dataset contains both important and unimportant features.
- **Cross-Validation**
  - **K-Fold Cross-Validation** ensures the model's stability by evaluating it on different subsets of the dataset, reducing the risk of overfitting.
- **Early Stopping (for Iterative Models)**
  - Stops training when the error on the validation set starts increasing, preventing the model from learning too much noise.
- **Data Augmentation (for Image and Text Data)**
  - Applies transformations such as rotation, flipping, resizing, etc., to artificially expand the training dataset.
- **Ensemble Learning**
  - **Bagging (Bootstrap Aggregating):** Trains multiple models on different subsets of the dataset to reduce errors.
  - **Boosting (e.g., XGBoost, AdaBoost):** Combines multiple models with different weights to improve performance.
- **Increase Training Data Size**
  - Collecting more data helps the model learn general patterns instead of memorizing specific examples.
- **Dropout (for Neural Networks)**
  - Randomly deactivates some neurons during training to prevent the model from relying too much on specific parts.

## 6.2. Apply overfitting avoidance methods to each algorithm

- **K-Nearest Neighbors (KNN):** To avoid overfitting in KNN, one common method is to increase the number of neighbors (*n_neighbors*). When *n_neighbors* is too small (e.g., *n_neighbors=1*), the model is prone to overfitting because it relies on a single point for classification, causing the model to learn too much detail from the training data. Increasing the value of *n_neighbors* helps the model generalize better, reducing the risk of overfitting and improving generalization when encountering new data.
- **Linear Regression:** In the case of Linear Regression, a method to avoid overfitting is to apply a small value for the regularization parameter C. The C parameter plays a crucial role in adjusting the level of regularization. When C is too large, the model is less regularized and more likely to overfit the training data. Conversely, reducing the value of C increases the level of regularization, which helps reduce overfitting and improves the model's generalization on test data.
- **Support Vector Machine (SVM):** In SVM, using a small value for C is a way to avoid overfitting. The C parameter in SVM controls the trade-off between maximizing the margin of separation and minimizing classification errors. When C is too large, the model can overfit the training data by fitting it too precisely.
- **K-Means Clustering:** For the K-Means algorithm, one method to avoid overfitting is adjusting the number of clusters (*n_clusters*). If the number of clusters is too large, the model may cluster the data too finely, leading to overfitting. Adjusting the number of clusters appropriately helps the model avoid learning too much detail and enables it to cluster more generally, avoiding overfitting.
- **Decision Tree:** In Decision Trees, to reduce overfitting, we can limit the depth of the tree (*max_depth*). When the depth is not limited, the tree can continue splitting until all data points are classified correctly, which can lead to overfitting.

## 6.3. Comparison of using and not using overfitting avoidance methods

- Using overfitting prevention methods helps improve the generalization ability of the model, reduce dependency on training data, and make the model more stable when encountering new data. Specifically:
    - When KNN has a large number of neighbors, the model generalizes better and reduces sensitivity to noise in the training data.
    - Using a small value for C in Linear Regression and SVM prevents the model from becoming too complex and allows it to handle new data points without overfitting.
    - K-Means with a reasonable number of clusters helps the model cluster data in a more general way, avoiding overly detailed clustering.
    - Limiting the depth of a Decision Tree helps prevent the decision tree from learning too specifically and ensures the model does not overly depend on the specific characteristics of the training data.

# REFERENCES


1. **Scikit-learn: Machine Learning in Python**, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.
2. **UCI Machine Learning Repository**: Online Shoppers Purchasing Intention Dataset.
3. **Gradient Descent:** https://viblo.asia/p/gradient-descent-giai-thich-tu-a-z-buoc-chan-quyet-dinh-cua-moi-mo-hinh-ai-pgjLNK2dV32
4. **Overfitting:** http://machinelearningcoban.com/2017/03/04/overfitting/
5. **Algorithms:** https://machinelearningcoban.com/
6. **Slides from instructor**